Assessing Information Security Risks in the Software Development Life Cycle

Douglas A. Ashbaugh
Software Engineering Services

Information is among the most important assets in any organization. Organizations are constantly building more complex applications to help them accomplish their mission and are entrusting their sensitive information assets to those applications. But are their information assets secure as they are transmitted, modified, stored, and displayed by those applications? Are new applications developed in a manner that will keep those sensitive information assets secure? How can we know for certain? The answers to these questions are all related and involve the process of assessing risk.

The concept of risk – *the net negative impact of the exercise of a vulnerability, considering both the probability and the impact of occurrence* [1] – is a concept that is hundreds of years old. Even the concepts of measuring and weighing *business risks* have been around for a long time. Insurance companies calculate risks every day and use the calculations to set rates for life, health, and property coverage.

Software development is a constant balancing act between functional requirements, funding, deadlines, limited resources, risk, and flexibility. Many of the current major software development life cycles (SDLCs) treat security simply as just one more non-functional requirement [2] and do not cover the topic of information security or address it in any detail. The result is often that security remains a non-functional requirement during the software development process. During the software engineering process, when resources, budgets, and schedules become tight, trade-offs must be made as some requirements must be dropped. This trade-off process introduces risk into the software development process. This is not to imply that security is always an important requirement of every software development effort. However, if confidentiality, integrity, and availability of the software or the information it stores, transmits, processes, or displays is important, then security should be considered an important requirement.

When risk is introduced into the software development process where confidentiality, integrity, and availability of the software or its information are important, then the result may be that the resulting software is not as secure as it needs to be. The General Accounting Office estimates $38 billion per year [3] in U.S. losses due to costs associated with computer software security lapses. How can we resolve this problem?

One solution is to apply information security risk assessment practices to the SDLC. Information security risk assessment is a practice used to ensure that computing networks and systems are secure. By applying these methods to the SDLC, we can actively reduce the number of known vulnerabilities in software as it is developed. For those vulnerabilities that we cannot or choose not to mitigate, we at least become aware of the risks involved as software development proceeds. The remainder of this article will focus on how to apply simple risk assessment techniques to the SDLC process.

Assessing Risk Within the SDLC

There are many different methodologies, tools, and techniques that can be used to assess risk. For the purposes of this article, we will focus primarily on a simple *qualitative* method. *Qualitative risk* assessments are all about identifying and relating risks relative to each other. The perceived impact of a loss associated with a risk is determined rather than the actual value associated with the loss. Also, because they are subjective in nature and do not require the precise knowledge required by other risk assessment methodologies, they typically take less time to conduct.

There are a number of essential elements of any qualitative risk assessment process. First, assets, threats, and vulnerabilities must be identified. An analysis can then take place that determines the likelihood of a vulnerability being exploited, the adverse impact of a vulnerability being exploited, and, finally, the level of risk associated with each threat-vulnerability pair. Controls may then be applied to eliminate or prevent the exercise of vulnerabilities. From a high-level view, the relationships between these entities are outlined in Figure 1. Looking at each of these elements a little closer, some of the ways we can apply them in each phase of the SDLC become visible.

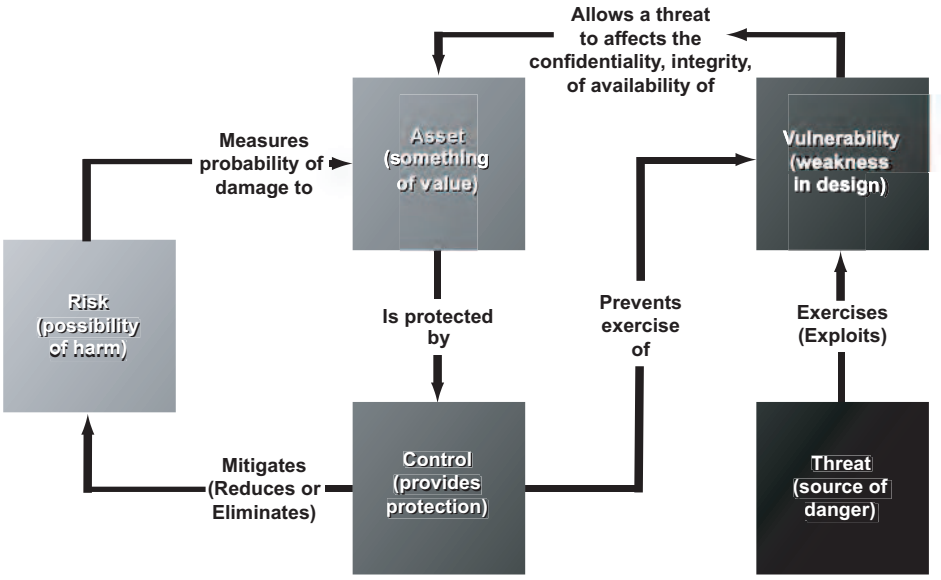
Identifying Threats

A *threat* is something that is a source of danger to an asset. Sources of threats may include (but are not limited to) those listed in Table 1 (see page 22).

The following organizations have published lists of potential threats:

- The National Institute of Standards and Technology (NIST) [4].

Figure 1: *Relationships Between Risk Analysis Elements*



Human Threats	Technical Threats	Environmental Threats
Data entry errors and omissions	Misrepresentation of identity	Electromagnetic interference
Inadvertent acts and carelessness	Unauthorized access to systems	Hazardous materials
Impersonation	Data contamination	Power fluctuations
User abuse and fraud	Malicious code	Water leaks
Theft, sabotage and vandalism	Session takeover	Fire
Espionage		Natural disasters

Table 1: *Threat Sources*

Checklist Sources	Application Code Scanning Tools	Network Vulnerability Scanners
The Open Web Application Security Project (OWASP) [8]	Paros Proxy	Nessus
Common Vulnerabilities and Exposures (CVE) [9]	WebScarab	Nmap
National Vulnerability Database (NVD) [10]	Web Inspect	Nikto
		Whisker

Table 2: *Identifying Vulnerabilities*

- The SysAdmin, Audit, Network, Security (SANS) Institute [5].
- The Center for Medicare and Medicaid Services (CMS) [6].

Many of these lists provide not only descriptions of threats, but also many real-world examples, as well as the potential impact a threat may have on the confidentiality, integrity, and availability of an asset.

It is important to identify threats because it is necessary to understand the potential impact a threat may have upon an asset. Once this potential impact is understood, controls that safeguard the asset from the threat can be identified. The earlier such controls are identified within the SDLC, the easier they can be incorporated into the design.

Identifying Vulnerabilities

A *vulnerability* refers to a weakness in design or controls that can be exploited by a threat to cause harm to an asset. According to the CERT Coordination Center, more than 90 percent of software security vulnerabilities are caused by known software defect types, and most software vulnerabilities arise from common causes. In fact, the top 10 causes account for about 75 percent of all vulnerabilities [7].

One of the easiest ways to identify vulnerabilities is to use checklists of common vulnerabilities developed by government and other interested agencies and groups. Another method would be to use application code scanning tools. Finally, network vulnerability scanners can also be used to find potential security vulnerabilities within an application as it exists in a network environment. A list of these checklists and tools is outlined in Table 2.

Identifying Assets

An *asset* is something of value to an organization. Assets associated with software development may include – but are not limited to – software, information, services, processes, functions, business rules, encryption keys, and methods.

It is important to identify assets as early as possible. Without knowing what assets need protection and without understanding what may happen when that protection fails, risk analysis techniques cannot produce worthwhile results [11].

In the case of identifying assets for software development, it is important to look to the business areas that will use the application being developed and its data. The end-users and their management are in the best position to understand what business information is essential to the mission and goal of the development effort. Furthermore, end-users and their management are in the best position to identify the business impact of the failure to protect the critical information.

Assets can also be identified by looking at policy. If an organization has good enterprise security policies, then assets can be discovered by a review of those policies to determine what types of assets the policy strives to protect. For example, a data classification policy might require that all customer information that could directly identify a customer (i.e. name, address, customer identification, etc.) must be considered confidential. If the application under development were to use or process any of this information, we would then consider that information an asset. Another example might include a disposal and reuse policy which requires that media must be thoroughly sanitized prior to its reuse or disposal. The assets in this case are hardware and/or reusable media storage devices and the information stored on them. Likewise, other policies may deal with the physical protection of people, hardware, documentation, buildings, and services.

Once information assets have been discovered, we can then apply analysis techniques to discover other potential assets. We can look at use cases, process flow diagrams, code, and other documentation and find where identified assets are accessed, read, written, modified, used, or monitored. Are specific processes, methods, services, functions, hardware, or individuals used to modify, display, transmit or store the assets? Any of these items may also be assets requiring protection.

Analyzing Risks

Risks occur when a threat exercises a vulnerability. The first step in analyzing risks is to determine the likelihood that a threat-vulnerability pair will be exercised. This is accomplished by consulting a likelihood table such as Table 3. Note that Tables 3-5 are provided as examples of what likelihood, impact, and risk tables might look like. Organizations such as NIST, SANS, CMS and others have developed a wide variety of tables that may be used in developing a table to fit your organization's specific needs.

Once the likelihood of a threat-vulnerability pair being exercised has been determined, then the impact to the assets identified if the threat-vulnerability pair is exercised may be determined according to Table 4.

Now the *risk* that a given threat may cause a specific impact by the exercise of a vulnerability may be determined. The *risk level* is determined by cross-referencing the likelihood with impact as shown in Table 5.

The process should be continued until all of the risks have been assessed. Once all risks have been assessed, management can prioritize, evaluate, and implement the most appropriate controls in order to mitigate the risks that have been uncovered. There are several options and strategies for mitigating risks including, but not limited to the following:

- **Risk assumption.** Choosing to accept the potential risk as is, or implementing controls to lower the risk to an acceptable level.
- **Risk avoidance.** Avoiding the risk by eliminating the cause or consequence of the risk.
- **Risk limitation.** Limiting the adverse affects of a risk by implementing additional controls.

- **Risk transference.** Transferring the risk to compensate for the loss, such as purchasing insurance.

Assessing Risks Within the SDLC

It is as important to know *when* to assess risks within the SDLC as it is to know *how* to assess risks within the SDLC. Risks should be assessed at the very beginning of the project and continue with each program review. It is also important to assess risks whenever requirements change and when the potential for new vulnerabilities and new threats are introduced.

Example of Risk Assessment

Consider an application for the financial services industry that requires a lot of personalized customer information such as names, employee identification numbers (which may be social security numbers), salaries, contributions to retirement plans, and dates of birth. Customer information for the application is gathered by a specific process and stored in a database. Complex business rules, which are proprietary to the organization, are used within the application to calculate individual retirement benefits. Furthermore, organizational policy requires that all information that could specifically identify an individual and all personal financial information about an individual must be considered confidential. In this example, the following would be the assets:

- Customer information specifically identifying an individual (name, identification number, date of birth).
- Customer financial information (salary, contributions to retirement plans).
- The database where the information is stored.
- The proprietary business rules.
- The process that gathers the data and stores it in a database.

The application allows employees of the organization to modify customer data. It also allows customers to view, but not modify or calculate, their retirement benefits online. Therefore, threats to such assets might include the following:

- Inadvertent, unauthorized modification of data (if customer data was modified or entered by mistake, the program could calculate incorrect benefits).
- Deliberate unauthorized exposure of the data (if a malicious person, either external or internal was able to break into the application).
- Deliberate unauthorized modification of the data (if a malicious individual or process was able to access the database, the data could be destroyed or corrupted).

Likelihood of Vulnerability being Exercised	Criteria Used to Determine Likelihood
High	• Threat source highly motivated and capable. • Controls preventing vulnerability ineffective or non-existent.
Medium	• Threat source highly motivated and capable. • Controls may prevent or at least impede vulnerability. OR • Threat source lacks motivation or capability. • Controls preventing vulnerability ineffective or non-existent.
Low	• Threat source lacks motivation or capability. • Controls prevent or significantly impede vulnerability.

Table 3: *Likelihood Determination* [12]

Impact	Criteria
High	• Costly loss of major tangible assets or resources. • May significantly violate, harm, or impede mission, reputation, or interest. • May result in human death or serious injury.
Medium	• Costly loss of assets or resources. • May violate, harm, or impede mission, reputation, or interest. • May result in human injury.
Low	• Loss of some assets or resources. • Noticeably affect mission, reputation, or interest.

Table 4: *Impact Determination* [12]

Next we need to identify potential vulnerabilities. Upon reviewing a list of the top 10 vulnerabilities from the OWASP site, we see that Structured Query Language (SQL) injection is a possible vulnerability for this application. If validation rules on input fields do not limit the ability to input malicious characters, such as apostrophes, and error-handling routines generate messages to the user that are not edited, then the possibility exists that a malicious user could see the name of the database and potentially some of the field names as well. Armed with this information, a malicious user could potentially alter, destroy, or disclose confidential customer data.

Let us examine the threat-vulnerability pair of a deliberate unauthorized modification of the data by an employee exercising the SQL injection vulnerability. To understand the potential problem, see Figure 2 (see page 24).

In this example, the application's control (code which could prevent the SQL injection from being exercised) is inadequate to protect the customer database (our asset) from the employee (our threat). If the threat were to enter the right character string into the application, the SQL injection vulnerability would reveal information about the customer database directly to the employee (our threat). Therefore risk exists that an employee (threat) could then use that knowledge (vulnerability) to directly alter, disclose, and/or destroy data in the customer database (our asset). But how great is this risk?

Based upon our criteria, the likelihood that this threat-vulnerability pair would be exercised is rated as *medium*. The reason that the likelihood is ranked at medium is that there are no controls in place to prevent the SQL injection from being exercised; however, the possibility that an employee (the only ones who can input

Table 5: *Risk Determination*

Threat Likelihood	Impact		
	Low	***Medium***	***High***
Low	Low Risk	Low Risk	Medium Risk
Medium	Low Risk	Medium Risk	Medium Risk
High	Medium Risk	Medium Risk	High Risk

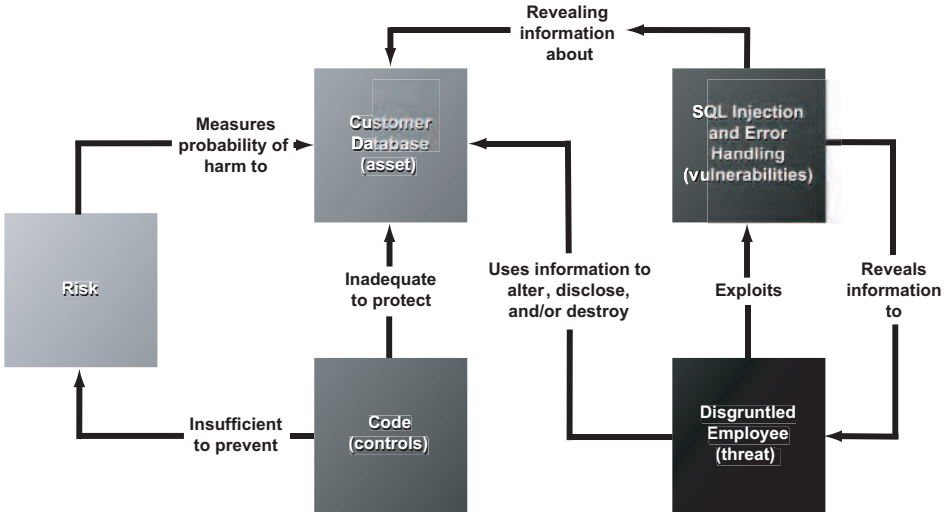


Figure 2: *Example – Risk Exists*

data in the application) is highly motivated and has the knowledge to actively seek out the database and impact it is negligible.

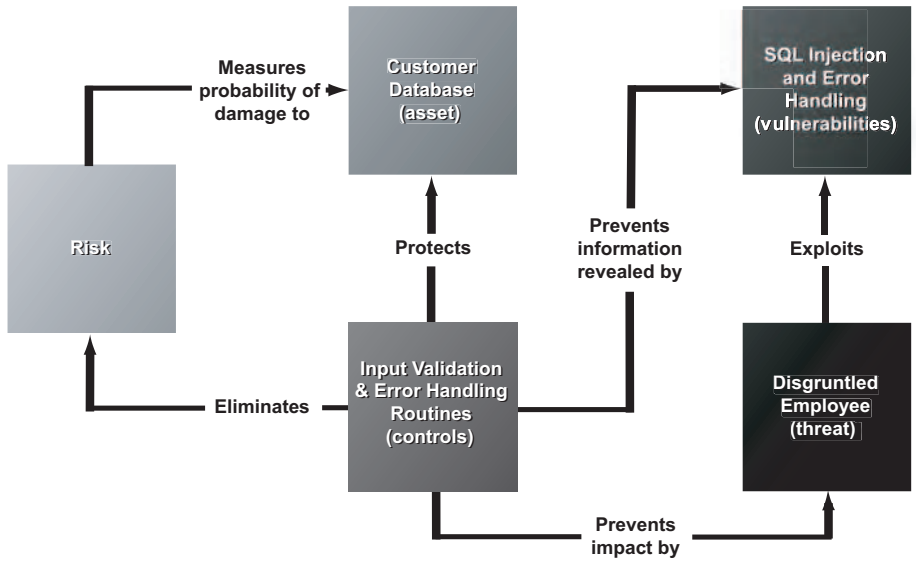
Next, we need to look at the potential impact of the threat-vulnerability pair being exercised. Our threat – the malicious user, through the SQL injection error – knows the name and fields within the database which contain confidential customer information. The malicious user could find customer salaries and social security numbers within this database and place them on the Web, affecting confidentiality. He could alter the information, affecting integrity. Or he could delete all of the information, affecting the availability of the data for others. In this case, we have to assume the worst: The malicious user would reveal the confidential information to others. Depending upon the customer, this could significantly violate the organization's reputation and mission. Therefore, the impact for this threat-vulnerability pair is rated high.

Cross-indexing the threat likelihood with the impact, we see that this is a medium risk. Management can now prioritize this risk alongside all other risks and determine what risk mitigation strategies may be appropriate.

Assume that management has decided to mitigate this risk by requiring that code be developed to prevent a SQL injection attack by requiring validation on all user inputs and only allowing certain error messages to reach end-users (see Figure 3).

In this instance, our control (the code to require proper input validation and error handling) prevents our threat (the employee) from exploiting the SQL injection vulnerability. The employee may attempt to enter the same character string as before, but the code that handles input validation would prevent him from either entering or executing a command using that character string. As a result, no information about the customer database is passed to the employee and he is unable to directly alter, disclose, and/or destroy data in the customer database.

Figure 3: *Example – Risk Mitigated*



Conclusion

Organizations continue to entrust precious assets to software that is developed with the same vulnerabilities over and over again. Ninety percent of software security vulnerabilities are caused by known software defect types, and the top 10 causes account for about 75 percent of all vulnerabilities. How can we reduce these vulnerabilities?

One answer is to apply the practice of assessing security risks within the SDLC. By assessing risks – common threats, vulnerabilities, and risks can be identified. Once the risks are known, then steps can be taken to eliminate or at least mitigate them. Unknown risks can not be eliminated or mitigated. That is why it is important we attempt to detect and analyze risks within the SDLC.◆

References

1. National Institute of Science and Technology. Risk Management Guide for Information Technology Systems. Special Paper 800-30. U.S. Department of Commerce. July 2002 <www.csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf>.
2. Petersen, Gunnar. "Phasing Security Into the SDLC – A Comparison of Approaches." Risk Management. 10 Jan. 2006 <http://1raindrop.typepad.com/1_raindrop/2006/01/phasing_securit.html>.
3. National Cyber Security Partnership. Improving Security Across the Software Development Lifecycle – Task Force Report. 1 Apr. 2004 <www.cyberpartnership.org/SDLCFULL.pdf>.
4. National Institute of Standards and Technology. Database of Threats and Countermeasures. 1999 <http://csrc.nist.gov/nissc/1999/proceeding/papers/o28.pdf>.
5. "@RISK: The Consensus Security Alert." SysAdmin, Audit, Network, Security (SANS) Institute. 5 June 2006 <www.sans.org/>.
6. "CMS Information Systems Threat Identification Resource." Centers for Medicaid and Medicare Services (CMS). 7 May 2002 <http://csrc.nist.gov/fasp/FASPDocs/risk-mgmt/Threat_ID_resource.pdf>.
7. National Cyber Security Partnership. Improving Security Across the Software Development Life Cycle – Task Force Report. 1 Apr. 2004

<www.cyberpartnership.org/SDLCFULL.pdf>.
8. Open Web Application Security Project (OWASP). "Top Ten Most Critical Web Application Security Vulnerabilities." <www.owasp.org/documentation/topten.html>.
9. The MITRE Corporation. "Common Vulnerabilities and Exposures." <http://cve.mitre.org/>.
10. National Institute of Standards and Technology. National Vulnerability Database <http://nvd.nist.gov/>.
11. Lavenhar, Steven, and Gunnar Petersen. "Architectural Risk Assessment." Cigital Inc. 2005 <https://buildsecurityin.us-cert.gov/portal/article/bestpractices/architectural_risk_analysis/architectural_risk_assessment.xml>.
12. National Institute of Standards and Technology (NIST). Risk Management Guide for Information Technology Systems. Special Paper 800-30, July 2002 <www.csrc.nist.gov/publications/nistpubs/800-30/sp80030.pdf>.

About the Author

Douglas A. Ashbaugh, CISSP, is a senior information security analyst for Enterprise Security Services where he provides information security analysis and remediation, policy and procedure development, and security awareness training to various clients. Ashbaugh has a Bachelor of Science in Engineering Operations from Iowa State University. He served eight years in the U.S. Air Force as an acquisition project officer and has worked as a software developer/analyst for the financial services industry.

Enterprise Security Services
1508 JF Kennedy DR STE 201
Bellevue, NE 68005
Phone: (402) 292-8660
Fax: (800) 660-5329
E-mail: dashbaugh@sessolutions.com

Letter to the Editor

Dear CrossTalk Editor,

In the March 2006 issue, Yuri Chernak, in his article *Understanding the Logic of System Testing*, outlined a possible methodological similarity between software testing and mathematical logic. I feel reality of software testing has a close resemblance to the way that experimental science works.

Some mathematical proofs developed in theory of software testing concerning the inexistence of a constructive criterion (i.e. algorithm) to derive the so-called Ideal Test Suite, (the silver bullet of system testing), state that the main goal of testing is restricted to show the presence of failures because testing can never completely establish the correctness of an arbitrary software system.

Skipping all formal aspects, these theoretical results are well summarized: Program testing can be used to show the presence of bugs, but never to show their absence!

Then the methodology of software testing is well rooted in modus tollens (Latin: mode that denies). Modus tollens is the formal name for contrapositive inference and can also be referred to as denying the consequent – it is a valid form of logical argument.

In fact, in experimental sciences, no number of positive outcomes at the level of experimental testing can confirm a theory, but a single counter-instance shows the scientific theory – from which the implication is derived – to be false.

Then, falsification with experimental implementation of modus tollens is just as essential to software testing as it is to scientific theories. Indeed, no number of passed test suites can prove the correctness of a program, but a single failed test suite shows the program to be incorrect.

Effective testing requires an empirical frame of reference rather than a theoretical one: Software reality is more about science than it is about computer science.

Software testing, for mathematical and theoretical reasons, is firmly set in the framework of experimental sciences and we need to see it in this perspective if we want to increase the comprehension of methodology and practice of software testing.

— Francesco Gagliardi
Department of Physical Sciences
University of Naples, Italy
francesco.gagliardi@na.infn.it